VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 Pavla Polová

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Pavla Polová**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: IDC CEMA, s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Tomáš Fabián, Ph.D.**

Konzultant bakalářské práce: Ing. Pavel Maneta

Datum zadání: 01.09.2018
Datum odevzdání: 30.04.2019

VYSOKÁ ŠKOLA BÁŇSKÁ - TECHNICKÁ UNIVERZITA OSTRAVA Fakulta elektrotechniky a informatiky 5 OSTRAVA

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 10. dubna 2019 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 10. dubna 2019

IDC CEMA s.r.o.
Malé náměstí 13, 110 00 Praha 1
IČ: 26482347 DIČ: CZ26482347

.....

Chtěla bych poděkovat kolegům ze svého týmu za spolupráci, trpělivost a příjemné pracovní prostředí. Mé poděkování patří také Ing. Pavlu Manetovi za odborné vedení a rady při zpracování této práce.

Abstrakt

Tato bakalářská práce pojednává o průběhu absolvování individuální odborné praxe ve firmě IDC CEMA, s.r.o. Popisuje zaměření firmy, úkoly, které byly zadány týmu, jehož jsem byla na praxi součástí, a také postupy při jejich řešení.

Klíčová slova: IDC, backend, Java, databáze, Angular

Abstract

This thesis describes the process of completing individual professional practice in the company IDC CEMA Ltd. It describes the focus of the company, tasks, that were given to the team I was part of during this practice, and the ways of solving them.

Key Words: IDC, backend, Java, database, Angular

Obsah

Seznam použitých zkratek a symbolů

ACID	–	Atomicity, Consistency, Isolation, Durability
CSS	–	Kaskádové styly
HTML	–	HyperText Markup Language
IDC	–	IDC CEMA, s.r.o.
MVC	–	Architektura MVC (Model - View - Controller)

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

VŠB-TU Ostrava nabízí studentům možnost absolvování individuální odborné praxe jako alternativu k vypracování bakalářské práce. Já jsem se pro tuto možnost rozhodla proto, že jsem chtěla nasbírat zkušenosti, které jsou v dnešní době čím dál cennější při hledání práce na technických pozicích, obzvláště pak v oboru informatika. Kromě toho jsem měla jedinečnou příležitost naučit se pracovat v týmu a vyzkoušet si, jak vypadá práce ve skutečné IT firmě.

O firmě IDC jsem se dozvěděla na kariérním veletrhu, který se konal na škole. Zaujaly mě nabízené pozice, jelikož jsem se zajímala o programování v jazyce Java. Byla jsem pozvána na pohovor, po jehož absolvování jsem zpracovala zadaný úkol a byla jsem přijata. Nastoupila jsem na pozici Application Developer do backendově orientovaného týmu. Po celou dobu mého působení v IDC jsem pracovala v Ostravských kancelářích na Technologické ulici v budově Trident. K práci mi byl poskytnut pracovní notebook.

Účelem této bakalářské práce je přiblížit zaměření firmy a popsat průběh mé odborné praxe. V první části své práce jsem představila samotnou firmu a také konkrétní projekty, na kterých jsem v průběhu praxe pracovala. Dále jsem popsala stěžejní technologie, se kterými jsem přišla do styku. V následujících kapitolách jsem pak uvedla jednotlivé zadané úkoly, popsala jsem postup při jejich řešení, a také jaký přínos měla implementace pro uživatele vyvíjeného software.

2 Zaměření firmy a pracovní zařazení studenta

Tato kapitola popisuje zaměření, kterému se věnuje firma IDC, a dále jednotlivých projektů, na kterých jsem během svého působení ve firmě pracovala.

2.1 O společnosti IDC

IDC, International Data Corporation, je přední analytická společnost. Poskytuje tržní zpravodajství a poradenské služby na trhu s informačními a komunikačními technologiemi a spotřební elektronikou. Více než 1000 jejích analytiků po celém světě sbírá a zpracovává data o prodejích počítačů, notebooků, tabletů, mobilních telefonů, chytrých hodinek, tiskáren a mnoha dalších typů produktů. Na základě těchto informací pak modelují a předpovídají budoucí vývoj IT trhu. Společnost byla založena v roce 1964 a poskytuje své služby společnostem ve více než 110 zemích [1].

2.2 Projekt Ferda

České IT týmy, mezi nimiž je i tým, ve kterém jsem během vykonávání praxe pracovala, pracují na vývoji a údržbě software určeného jak pro analytiky, kterým umožňuje vkládat a upravovat data, tak pro samotné zákazníky, kteří data kupují. Já osobně jsem byla součástí týmu Ferda Server. Jedná se o tým, který vyvíjí backendovou část hlavní aplikace Ferda.

Ferda je interní desktopová aplikace, která slouží analytikům pro potřeby jejich výzkumu. Analytici do ní denně vkládají velké množství dat a různě s nimi manipulují. Ferda jim poskytuje širokou škálu možností úpravy, vizualizace, importu a exportu dat. Výsledkem práce analytika je potom *modul* (neboli *technologie*) - souhrn velkého množství dat o prodejích konkrétního druhu výrobku. Moduly se dělí na dva typy. Historické moduly popisují přehled prodeje do aktuálního data, příkladem takového modulu může být například PC Hist, neboli historie prodeje počítačů. Kromě historických modulů existují také forecast moduly. V takových modulech analytici odhadují a modelují vývoj trhu v budoucnu.

V současnosti je k dispozici nová verze klienta Ferda NG, nicméně některá funkcionalita zatím zůstává ve starém desktopovém klientovi. Náš tým kromě samotného Ferdy vyvinul také několik dalších aplikací, které umožnují Ferdu spravovat a kontrolovat jeho stav. Těmi, se kterými jsem se během praxe setkala, jsou Ferda Admin a Ferda Monitor. Ve své práci budu popisovat různé úkony, které jsem spolu se svým týmem vykonávala jak na backendové části samotného Ferdy, tak predevším na těchto dvou aplikacích, abychom výslednou aplikaci udělali rychlejší a stabilnější a abychom analytikům zpříjemnili a zjednodušili práci.

2.3 Projekt Ferda Monitor

Ferda Monitor je webová služba, která v pravidelných intervalech monitoruje stav aplikace Ferda a jejích komponent. Uživatelé i vývojáři zde mohou zobrazit spoustu užitečných informací, které

jim pomáhají porozumět aktuálnímu stavu Ferdy, a v případě potřeby rychleji odhalit problémy, kvůli kterým Ferda nefunguje správně. Mezi hlavní položky, které tento nástroj sleduje, mimo jiné patří serverové instance a jejich stav, instance builderů (např. Excel builder, sloužící k vytvoření dokumentu v Excelu na základě předem definované šablony a následnému doručení zákazníkům) a jejich stav a databázové checky (pravidelně spouštěné dotazy nad databází) a informace o tom, kdy naposledy proběhly a jaký byl jejich výsledek.

V závislosti na zjištěných informacích pak Ferda Monitor rozlišuje 3 stavy:

1. Aplikace běží bez problémů.
2. Aplikace běží, ale došlo k určitým problémům, které mohou ovlivnit funkčnost.
3. Došlo ke kritickým problémům, kvůli kterým aplikace nemůže fungovat.

2.4 Projekt Ferda Admin

Ferda Admin je webová aplikace psaná v Angularu, která umožňuje kompletní definici a správu metadat. Poskytuje funkcionalitu zvanou Self Service, v rámci níž si uživatelé mohou sami sledovat a spravovat moduly.

Většina zadaných úkolů během mé praxe se týkala přepisu funkcionality Self Service, která byla dostupná pouze ve staré verzi desktopového klienta, do Ferdy Admina, aby uživatelé již starého klienta nepotřebovali a mohl být co nejdříve definitivně vypnut.

3 Použité technologie a nástroje

V této kapitole se budu věnovat stručnému popisu klíčových technologií, se kterými jsem pracovala v průběhu praxe.

3.1 Java

Java je objektově orientovaný programovací jazyk. Byl vyvinut v roce 1995 společností Sun Microsystems. Java je navržená tak, aby byla pro vývojáře jednoduchá, aby poskytovala bezpečnost a robustnost a aby byla přenositelná (nezávislá na architektuře). Jedná se o interpretovaný jazyk, což znamená, že se aplikace místo do strojového kódu obvykle překládají do Java bytecode, který může být následně spuštěn na jakémkoli zařízení, na kterém je k dispozici tzv. *interpret*, neboli virtuální stroj Javy (JVM - Java Virtual Machine). Díky tomu jsou vytvořené programy multiplatformní. Podle společnosti TIOBE Index jde o nejoblíbenější programovací jazyk na světě [2][3].

3.2 Angular

Angular je framework, který slouží k vývoji webových aplikací. Je založený na jazyce TypeScript. Někdy bývá také nazýván *Angular 2+*, jelikož se jedná o přepracovanou verzi původního AngularJS založeném na JavaScriptu. Tato nová verze Angularu, která vznikla v roce 2016, se oproti původnímu AngularJS založeném na architektuře MVC vyznačuje použitím hierarchie *komponent*, což jsou základní prvky uživatelského rozhraní [4][5].

3.3 PostgreSQL

PostgreSQL je objektově-relační databázový systém. Jedná se o open source software. Byl primárně navržen pro unixové platformy, nicméně je přenositelný a použitelný i na ostatních platformách, jako je Windows nebo Mac OS. Poskytuje pokročilé databázové nástroje, jako jsou uživatelsky definované typy, dědičnosti tabulek, pohledy, indexy a splňuje stoprocentně zásady ACID [6][7].

3.4 Git

Git je systém pro správu verzí, který byl vyvinut Linusem Torvaldsem v roce 2005. Jedná se o distribuovaný systém, což znamená, že každý vývojář má celý repozitář stáhnutý lokálně na svém zařízení. Slouží pro kolektivní vývoj rozsáhlých projektů, především nelineárním způsobem. K tomu slouží tzv. větve, díky kterým je možné se odklonit od hlavní linie a pracovat, aniž by do hlavní linie bylo zasahováno. Po dokončení práce je možné větve opět sloučit. Nejmenší jednotkou práce je commit, který by měl oddělovat každou podstatnou provedenou změnu [8][9].

4 Zadané úkoly

Jak již bylo zmíněno, hlavním úkolem, kterého jsme se po většinu času drželi, byl přesun funkcionality z původního desktopového klienta do aplikace Ferda Admin. V době, kdy jsem do týmu nastoupila, byl Ferda Admin funkční, nicméně v rámci Self Service byly po zvolení konkrétního modulu dostupné pouze následující záložky:

1. Home - zde mohou uživatelé vidět základní informace o modulu (název, ID, vlastník apod.), a dále je možné menit některé vlastnosti modulu, například změnit časové období pro daný modul (Current timescale), kterým se rozumí časový rámec, pro který analytici aktuálně sbírají data. V závislosti na nastavení se může jednat o modul kvartální, ve kterém se data klientům doručují po kvartálech (např. historie prodeje počítačů v České republice za 2018Q4), je také možné nastavit doručování po pololetích nebo celém roce. Součástí karty Home je také seznam nedávno provedených aktivit.

2. Overview - na této kartě si uživatel může prohlédnout *dimenzní mapy* a *model management*. Dimenzní mapy obsahují seznamy hodnot sloužících ke kategorizaci a třídění dat. Příkladem může být Geography map (geografická mapa). Ta obsahuje všechny existující lokality, které mohou být použity v rámci georafického určení. Na nejvyšší úrovni jsou hlavní regiony, jako EMEA (Evropa, Střední východ a Afrika). Na nižších úrovních se pak nachází jednotlivé státy, provincie a města. Například atribut Country (země) pochází právě z této dimenzní mapy. Model management je dimenzní mapa obsahující souhrn všech parametrů, které definují konkrétní model výrobku. Dále se na této kartě nacházel zjednodušený seznam atributů rozdělených dle kategorií.

3. Data Audit - zde si může uživatel zobrazit seznam změn v datech a různě tyto změny filtrovat. Podle nastavení konkrétního modulu se může jednat buď o podrobný výpis změn včetně změněných hodnot, nebo pouze o záznam o druhu provedené aktivity.

4. Remove - po jednoduchém potvrzení může uživatel s příslušným oprávněním modul smazat.

Během mého působení postupně docházelo k vytváření nových záložek a rozšiřování stávajících o nové funkce.

4.1 Seznam atributů

Na kartě Overview již nebylo dostačující zobrazovat stávající tabulky se seznamy atributů dle jejich kategorií. Bylo potřeba nahradit je tabulkou sjednocující veškeré atributy daného modulu, která bude navíc obsahovat jejich vlastnosti:

1. **ID Atributu**
2. **Název atributu**
3. **Source** (zdroj) - informace o tom, odkud daný atribut pochází. Může se jednat o atribut z nějaké dimenzní mapy nebo o atribut derivovaný, což znamená, že je jeho hodnota odvozena z hodnot jiných atributů na tomto modulu. Pokud hodnota zdroje není vyplněna, znamená to, že se jedná čistě o datový atribut.
4. **Source detail** (informace o zdroji) - doplňující informace o zdroji, ze kterého pochází atribut. V tomto sloupci se nachází název dimenzní mapy, jedná-li se o dimenzní atribut, nebo v případě derivovaného atributu o specifikaci jeho typu.
5. **Visibility** (viditelnost) - informace o tom, jestli je atribut viditelný pro uživatele.
6. **Display** (zobrazení) - informace, zda se jedná o hlavní atribut dané úrovně dimenzní mapy.
7. **Mandatory** (povinnost vyplnění)
8. **Read Only** (pouze pro čtení) - informace, jestli je možno upravovat hodnotu tohoto atributu.
9. **Type** (datový typ atributu)

Zároveň mělo být umožněno změnit název atributu nebo přepnout jeho viditelnost.

Na stránku byla přidána klasická HTML tabulka, která byla naplněna atributy a jejich vlastnostmi. Některé vlastnosti atributů (například název) byly obsaženy přímo ve třídě atributu, takže bylo jednoduché je získat. U některých vlastností, jako jsou právě dimenzní mapy, bylo už k získání potřeba složitějších metod. Proto byla vytvořena třída `AttributeWrapper`, která shromažďuje všechny výše uvedené informace o daném atributu (viz výpis 1).

```
for (let attr of this.allAttributes) {
   let attribute: AttributeWrapper = new AttributeWrapper();
   attribute.attribute = attr;
   attribute.source = this.getSource(attr);
   attribute.sourceDetail = this.getSourceDetail(attr);
   attribute.visibility = this.getVisibility(attr);
   attribute.displayIcon = this.getDisplayIcon(attr);
   attribute.mandatoryIcon = this.getMandatoryIcon(attr);
   attribute.readOnlyIcon = this.getReadOnlyIcon(attr);
   attribute.type = this.getType(attr);
   this.tableAttributes.push(attribute);
}
```

Výpis 1: Vytváření objektu AttributeWrapper

V uvedeném cyklu se postupně provolávají metody, které získávají informace o daném atributu a ukládají je do nové proměnné. Na závěr se takto nově vytvořený objekt `AttributeWrapper` uloží do seznamu všech atributů, které se mají zobrazit v tabulce.

U sloupců, kde jedinými možnými hodnotami bylo true a false (a případně null), je zobrazena ikona ✓ pro hodnotu true nebo žádná ikona v ostatních případech. K tomu jsem využila knihovnu Font Awesome. U každé takové buňky se dynamicky mění CSS třída (viz výpis 2).

```
<!-- Viditelnost -->
<td width="80px" align="center"><i class="{{attr.displayIcon}}"></i></td>
<!-- Povinnost -->
<td width="85px" align="center"><i class="{{attr.mandatoryIcon}}"></i></td>
<!-- Read Only -->
<td width="100px" align="center"><i class="{{attr.readOnlyIcon}}"></i></td>
```

Výpis 2: Zobrazování ikon v tabulce

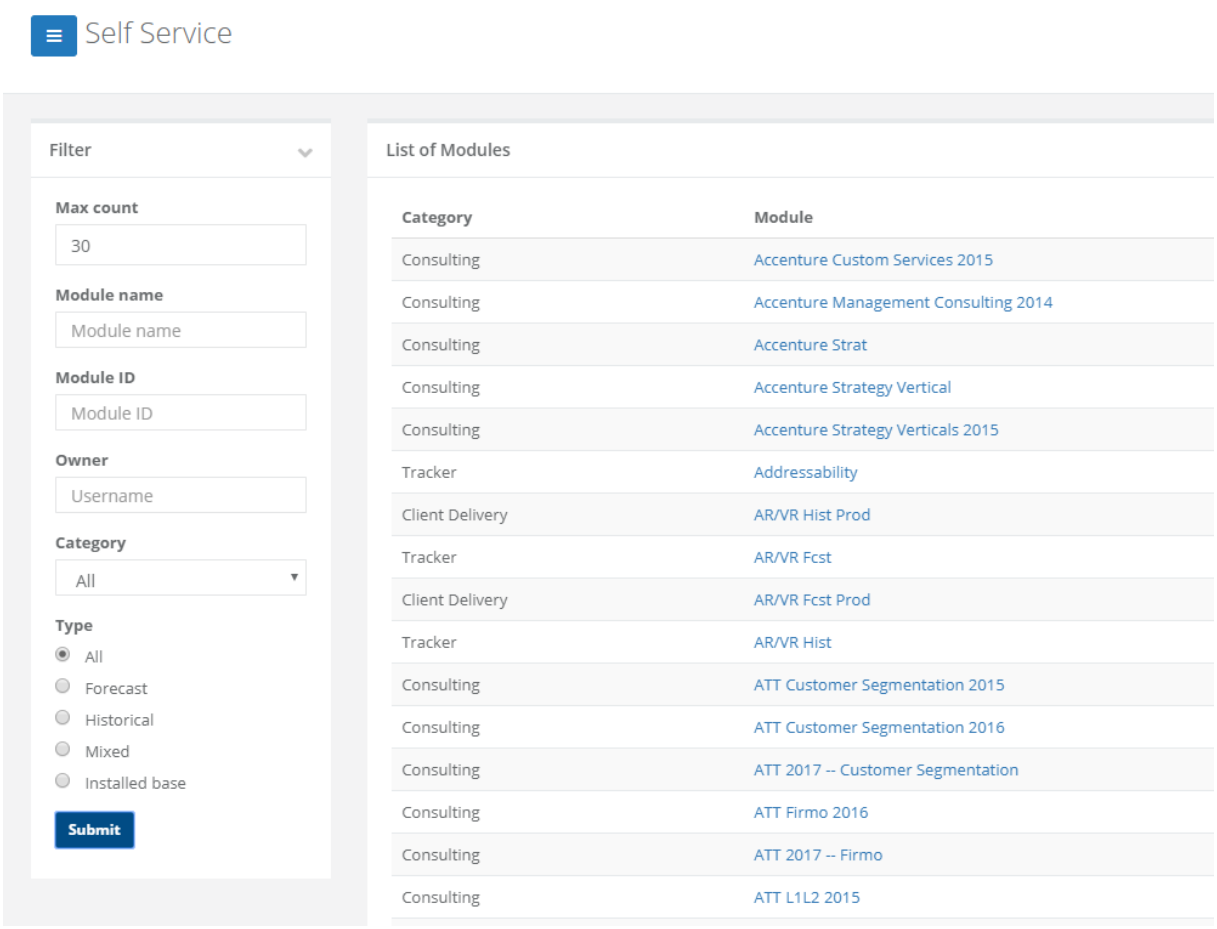
V objektu `AttributeWrapper` pak nejsou informace například o viditelnosti uloženy jako true/false. Místo toho metoda `getDisplayIcon` vrací přímo řetězec s názvem třídy, která se má použít pro zobrazení ikony.

Pro úpravu atributu byla vytvořena metoda, které se předávají nové vlastnosti (například nový název). V rámci této metody se volá request `SaveOrUpdateRequest`, který nové vlastnosti atributu ukládá do databáze. Tomu samozřejmě musí předcházet kontroly, zda jsou nové informace validní (tedy jestli není například název atributu prázdný nebo neobsahuje nepovolené znaky), a také jestli má uživatel oprávnění k měnění těchto informací.

4.2 Nové tabulky

V aplikaci Ferda Admin se na různých stránkách nachází tabulky, které obsahují například seznam modulů (technologií) nebo seznam atributů daného modulu a jejich vlastností (viz sekce 4.1). Pro zjednodušení dalšího rozšiřování těchto tabulek bylo rozhodnuto o přechodu na novou komponentu, kterou je Turbo Table z knihovny PrimeNG. Ta poskytuje podporu pro mnoho různých funkcí, z nichž bylo nejpodstatnější stránkování a globální filtrování výsledků (vyhledávání napříč všemi sloupci).

Mým úkolem bylo nahradit obyčejnou HTML tabulku na stránce Self Service, kde se nachází seznam modulů. Tam se moduly získávají dynamicky ze serveru. Práce byla tedy komplikovanější, protože muselo být v rámci Lazy Loadingu ručně naimplementováno vlastní filtrování a stránkování. Druhou komplikací bylo, že na stránce Self Service již byl jeden filtr, který umožňoval filtrovat výsledky zadáním jména, id nebo vlastníka modulu, nebo také vybrat kategorii a typ (viz obrázek 1). Filtrované výsledky pak měly být průnikem obou těchto filtrů.

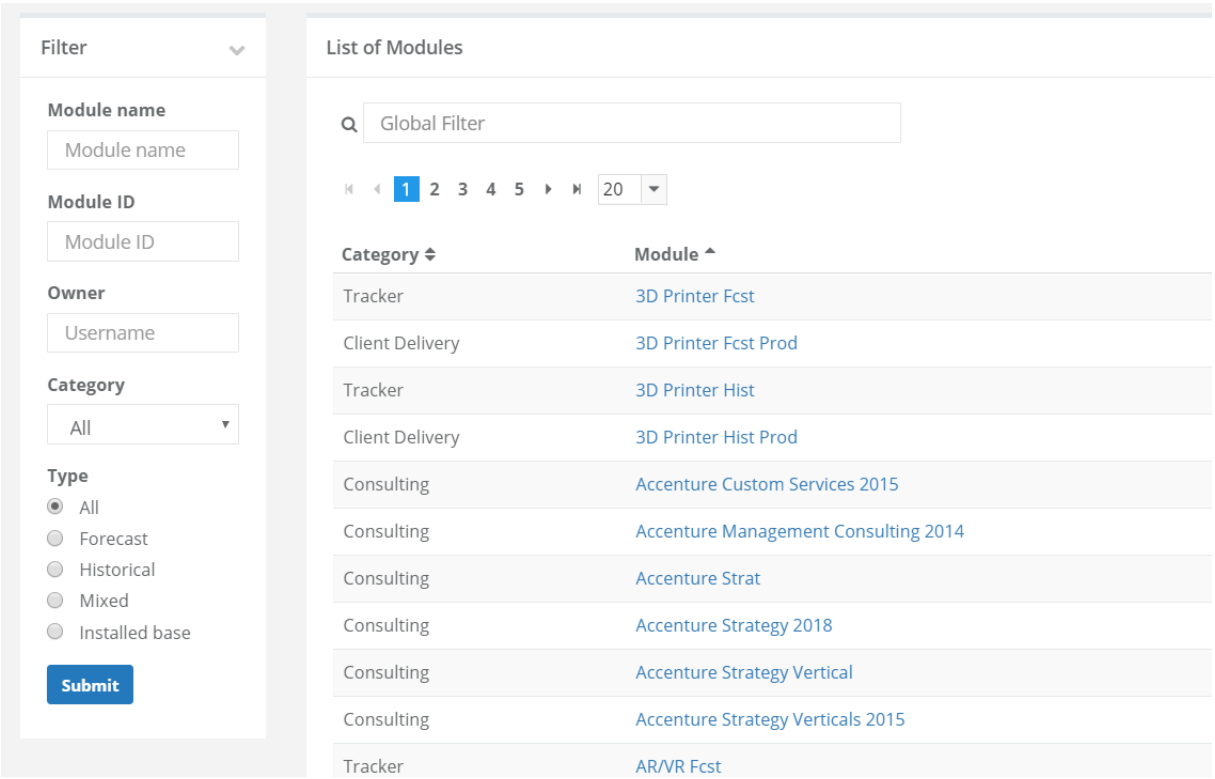


Obrázek 1: Původní seznam modulů

Pro Lazy Load filtrování byla přidána nová metoda `loadModulesLazy`, jejímž parametrem

je `LazyLoadEvent`. V něm se předávají všechny potřebné informace o aktuálním nastavení tabulky: počet řádků na stránku, offset (čili aktuální stránka), sortField (sloupec, podle kterého se tabulka aktuálně abecedně řadí), zda se řadí sestupně či vzestupně a také vyhledávací řetězec, který je vložen v globálním filtru. Tyto údaje se používají pro vytvoření requestu na server, který vrací vždy pouze tolik modulů, kolik se aktuálně má zobrazovat na stránce, a samozřejmě s požadovaným filtrováním. Tato metoda se pak volá při každé změně v tabulce - při přepnutí stránky ve stránkování, při změně řetězce v globálním filtru nebo při změně počtu řádků na stránku. Tím je zaručeno, že se moduly vždy správně aktualizují.

Původní filtr, který na stránce již byl, má tlačítko Submit, kterým se potvrzuje výběr vyhledávaných výrazů a následně se odesílá request na server. Toho jsem využila. Po kliknutí na tlačítko Submit v tomto filtru se zavolá metoda `loadModulesLazy` s uměle vytvořeným parametrem `LazyLoadEvent`, do kterého jsou vloženy informace získané z tohoto filtru v požadovaném formátu. Tyto informace jsou uložené v poli a aktualizují se při každém zavolání metody `loadModulesLazy`, aby vždy došlo k uložení aktuálního nastavení. Tabulka se tedy správně aktualizuje jak při filtrování přes integrovaný filtr komponenty Turbo Table, tak i při použití druhého, externího filtru. Vzhled tabulky se stránkováním a filtrem je zobrazen na obrázku 2.



Obrázek 2: Nový seznam modulů

4.3 Usage Logging

Stejně jako sledování změn v datech bylo potřeba také monitorovat aktivitu uživatelů ve Ferda Admin. Cílem bylo ukládat do databázové tabulky informace o všech podstatných provedených změnách na modulu.

Pro logování provedených akcí existuje již speciální modul **Data Collector**, který ukládá tato data. Používal se pouze pro logování v rámci samotné aplikace Ferda, nicméně nyní bylo potřeba zavést toto logování také pro akce provedené ve Ferda Admin.

Logování mělo probíhat při provedení obvyklých operací, například zobrazení seznamu modulů, zobrazení detailu konkrétního modulu, změna názvu atributu, změna nějakého parametru modulu, nebo v případě, že dojde k chybě (výjimce). V každém záznamu by pak měly být obsaženy informace o uživateli (jméno, IP adresa, země), podrobné informace o provedené akci a také čas provedení.

Moje práce byla částečně usnadněna tím, že jsem mohla použít některé služby, které již byly vytvořeny v rámci jiného projektu. Konkrétně se jednalo o již připravenou službu, která zjišťuje uživatelské jméno a IP adresu, a také službu, která posílá request pro uložení dat do Data Collectoru.

Na všechna potřebná místa v kódu bylo do metod přidáno zavolání metody `report` definované na nově vytvořeném handleru `UsageCollectorHandler`. Tato metoda přijímá následující parametry:

1. featureDetailOne - hlavní kategorie provedené akce. Mohlo se jednat například o „Self Service“ (stránku s moduly) nebo „Scheduled jobs“ (stránku zobrazující asynchronně prováděné úlohy).

2. success - boolean, informující o tom, zda se pokus o provedení akce zdařil.

3. featureDetailTwo - druhá úroveň popisu provedené akce. Mohla obsahovat například „Technology Detail“ v případě kliknutí na detail konkrétního modulu.

4. featureDetailThree - dodatečná informace, využívaná například pro zalogování akce vyhledávání ve filtru.

5. technologyId - id modulu, vyplněné pouze v případě, že se akce vztahovala k nějakému konkrétnímu modulu (například při měnění názvu atributu na tomto modulu).

6. customAttributes - nepovinný parametr obsahující pole objektů, které mají dále specifikovat akci. Tento parametr byl využit při chybě pro zalogování výjimky, ke které došlo.

Jelikož kromě tohoto logování již probíhalo logování podobných informací také do Google-Analytics, byla vytvořena služba `userActionLoggingService`, která zajišťuje obojí logování současně. Příklad zalogování změny názvu atributu je uveden ve výpisu 3.

```
this.userActionsLoggingService.log("Self Service", "Attribute Caption
   Change", "Attribute Caption Change", null, this.technology.id);
```

Výpis 3: Volání metody log při změně názvu atributu

Samotná metoda `log` v `userActionsLoggingService` je ukázána ve výpisu 4.

```
public log(category: string, featureDetailTwo: string, featureDetailThree?:
    string, analyticsLabel?: string, actionSuccessful?: boolean,
    technologyId?: string, customAttributes: Object = {}): void {
        if (AppSettings.IS_ENABLED_LOGGING_TO_USAGE_COLLECTOR) {
            this.usageCollector.report(category, actionSuccessful,
                featureDetailTwo, featureDetailThree, technologyId,
                customAttributes);
        }
}
```

Výpis 4: Metoda log

Díky logování aktivit ve Ferda Admin je nyní jednodušší dohledat historii provedených změn. Navíc, pokud dojde k chybě, je možno snadno dohledat konkrétní chybovou hlášku.

4.4 HLC transformace

Pro usnadnění práce analytiků byl zaveden nový typ transformace: *HLC transformace* (high level change - změna na vysoké úrovni).

Transformace obecně slouží k přenosu dat. V současnosti existuje několik druhů transformací:

1. jednoduché transformace, ve kterých dochází k přesunu všech vybraných dat. Tato transformace nejprve data odstraní na cílovém modulu, a poté do něj přesune data z modulu zdrojového.

2. tzv. topline transformace, které přenáší pouze elementární informace o prodeji (tzv. *shipmenty*). Tento typ transformace zachovává procentuální rozdělení číselných hodnot na shipmentech podle určitých kategorií (tzv. *splity*). Topline transformace data na cílovém modulu neodstraňuje, pouze aktualizuje stávající data a přidává nová.

3. HLC transformace.

HLC transformace umožňují změnit celkovou sumu nějakých číselných hodnot (dále *figures*), například počtu prodaných kusů, u určité skupiny shipmentů. Nejpodstatnější vlastností HLC transformací je jejich schopnost zachovat původní poměry mezi shipmenty. Při této transformaci

uživatel definuje skupinu shipmentů pomocí filtru a transformace pak přerozdělí hodnoty mezi tyto shipmenty tak, aby byl zachován původní poměr mezi nimi a aby celkový součet dával požadovanou hodnotu.

Tabulka 1: Zdrojový a cílový modul HLC transformace

Zdroj				**Cíl**			
Country	Quarter	Product	Units	Country	Quarter	Product	Units
USA	2018Q1	Desktop	123	USA	2018Q1	Desktop	100
USA	2018Q1	Notebook	12	USA	2018Q1	Notebook	100
USA	2018Q1	Workstation	442	USA	2018Q1	Workstation	100
USA	2018Q1	Ultra Portable	435	USA	2018Q1	Ultra Portable	100
USA	2018Q1	Mobile Workstation	435	–	–	–	–
–	–	–	–	USA	2018Q1	Ultra Slim	100

Tabulka 2: Výsledek HLC transformace

Výsledek				
Country	Quarter	Product	Units	
USA	2018Q1	Desktop	123	
USA	2018Q1	Notebook	12	
USA	2018Q1	Workstation	442	
USA	2018Q1	Ultra Portable	435	
–	–	–	–	**Error**
USA	2018Q1	Ultra Slim	0	**Set to 0**

Mým úkolem bylo vyřešit problematiku chybějících řádků. V tabulce 1 je uveden příklad zdrojového a cílového modulu. Oba tyto moduly sdílí atribut Product. Poslední dva řádky ukazují specifické případy, kdy má zdrojový a cílový modul řádek navíc. Pokud obsahuje zdrojový modul řádek, který na cílovém modulu chybí (v uvedeném příkladu je to Mobile Workstation), dojde u daného řádku k chybě, protože transformace v takovém případě nemůže být správně provedena. Pokud však přebývá řádek na straně cílového modulu, transformace se může provést, ale počet Units na tomto řádku musí být vynulován (viz tabulka 2).

Provádění jakýchkoli transformací funguje tak, že se ze zdrojového modulu vytvoří dočasná tabulka s daty, které se mají přenést na modul cílový. Tato tabulka je pak předávána v jednotlivých krocích řetězce do různých tříd, které s daty manipulují, validují je a předávají je dál. Po provedení posledního kroku by data v této tabulce měla být správně přetransformována a připravena pro vložení do cílového modulu. Mým úkolem bylo přidat do řetězce další mezikrok, který z cílového modulu získá shipmenty, které nejsou obsaženy ve zdrojovém modulu, a vloží

je do této dočasné tabulky s nulovými figures. SQL dotaz pro zjednodušený příklad uvedený v tabulkách 1 a 2 je uveden v následujícím výpisu 5.

```
SELECT tt.id, tt.country, tt.quarter, tt.product, 0 AS units
FROM target_technology tt
LEFT JOIN source_technology st
ON st.id = tt.id AND st.country = tt.country AND st.quarter = tt.quarter AND
    st.product = tt.product
WHERE st.id IS NULL
```

Výpis 5: Získání chybějících shipmentů

Každá transformace má v metadatech uloženou svou konfiguraci, ve které jsou specifikována její nastavení. Především se jedná o zdrojový a cílový modul, a také mapování mezi zdrojovými a cílovými atributy. Nejprve bylo potřeba získat názvy všech atributů ze zdrojového a z cílového modulu, jelikož jejich názvy se mohou lišit, a také názvy všech přítomných figures, které se mají vložit s hodnotou 0. Z těchto atributů a figures byl sestaven dotaz ve výpisu 6.

```
// vytvoření řetězce sloupců pro dotaz SELECT (např. tt.coutry, tt.quarter,
    tt.product,...) se všemi atributy
String selectAttributes = attributesList.stream()
   .map(attribute -> "tt." + attribute.getTargetAttribute().getId() + " AS "
       + attribute.getSourceAttribute().getId())
   .collect(Collectors.joining(", "));

// vytvoření řetězce pro vynulované figures (např. 0 AS units, 0 AS
    user_values)
String selectFigures = figuresList.stream()
   .map(figure -> "0 AS " + figure.getFigure().getId())
   .collect(Collectors.joining(", "));
```

Výpis 6: Utvoření dotazu

Takto sestavené řetězce se pak používají ve finálním dotazu i s názvy příslušných tabulek. Výsledek tohoto dotazu je zároveň vložen do uvedené tabulky, která je již výsledkem tohoto kroku řetězce.

Před zavedením HLC transformací museli analytici nejprve vyexportovat data, upravit je, a následně znovu naimportovat, což bylo zvlášť v případě větších objemů dat velmi časově náročné. Díky novému způsobu je pro ně tento proces jednodušší a rychlejší. Po implementaci na serveru byla do aplikace Ferda Admin pro uživatele přidána možnost tyto transformace vytvářet, upravovat a spouštět (viz sekce 4.5).

4.5 Seznam transformací

Jako reakci na požadavek analytičky byla do Ferdy Admina přidána nová záložka: Transformations. Ta měla obsahovat seznam *transformací*. Transformace jsou funkce, které slouží k přenosu dat napříč moduly. Součástí této karty jsou nyní 3 tabulky:

1. transformace, ve kterých tento modul figuruje jako zdroj, neboli transformace, u nichž dochází k přenosu dat z tohoto modulu na jiný modul,
2. transformace, v nichž je tento modul cílový,
3. historie spouštění těchto transformací.

Tyto tabulky obsahují základní informace o transformacích, jako je jejich název, zdrojový a cílový modul a typ. Informace se načítají z databáze pomocí requestů poslaných na server. První dvě tabulky obvykle obsahují menší počet záznamů, avšak u poslední tabulky s historií spouštění transformací může být záznamů velmi mnoho. Proto bylo u této tabulky implementováno stránkování a lazy loading za použití komponenty Turbo Table, viz sekce 4.2.

Kromě zobrazení existujících transformací a historie jejich spouštění bylo pak třeba implementovat také vytváření nových HLC transformací. Aby došlo k zachování funkcionality, na kterou jsou uživatelé zvyklí ze stávající Self Service ve staré verzi klienta, pokusili jsme se co nejvíce přiblížit tomu, jak vytváření transformací fungovalo právě tam. Mým úkolem bylo do nového modálního okna, které se zobrazilo po kliknutí na tlačítko pro přidání transformace, přidat sekci General Information (základní informace).

Tato sekce zahrnuje 5 různých vstupů - zdrojový modul, zdrojový snapshot, period type, název transformace a komentář. Viz obrázek 3

Source Technology - v této sekci vybírá uživatel zdrojový modul z dropdownu obsahující seznam všech existujících modulů. Jedná se o modul, ze které se budou přenášet data do Target Technology (cílového modulu), kterým je právě ten modul, na kterém uživatel transformaci vytváří. Byla vytvořena metoda `loadTechnologies`, v rámci níž se pomocí requestu získávají ID a názvy všech existujících modulů a ukládají se do seznamu. Tento seznam je pro ušetření času a počtu requestů ukládán do session storage. Metoda `loadTechnologies` se volá při každém otevření modálního okna, ale pouze v případě, že objekt `technologyList` v session storage je nastaven na defaultní hodnotu null, čili moduly ještě nebyly načteny.

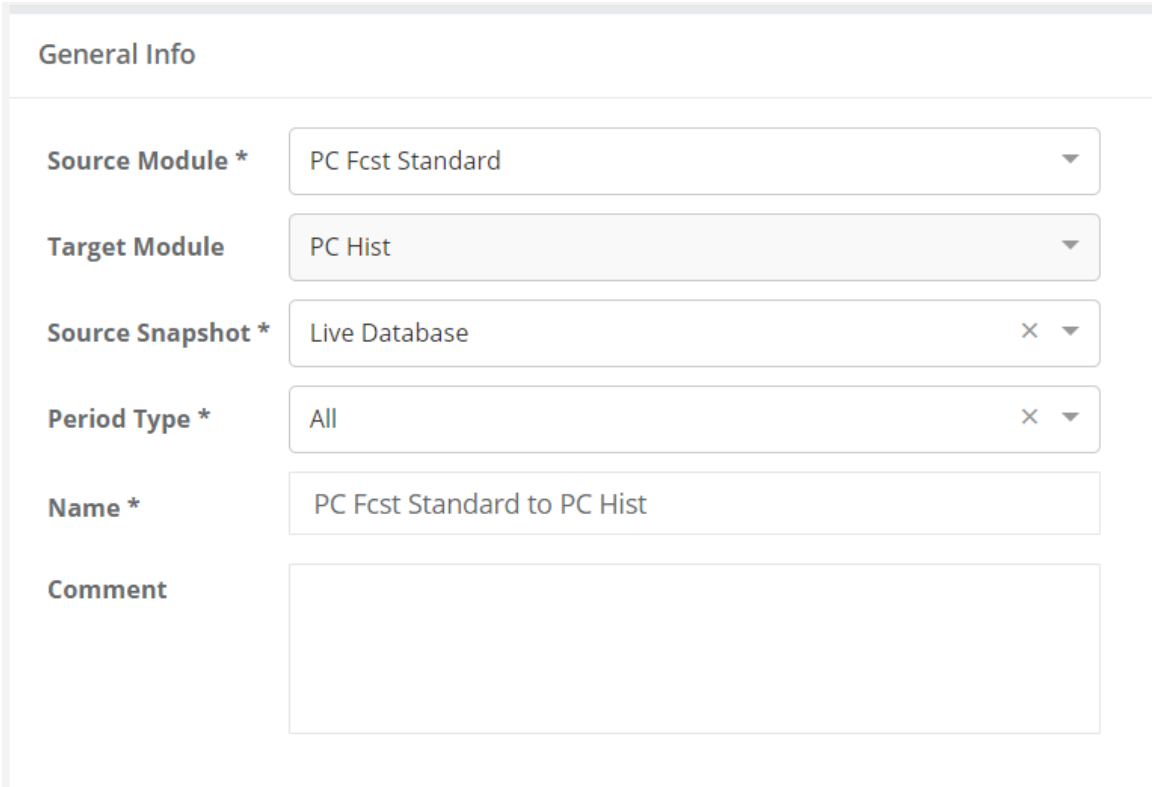
Po zvolení zdrojového modulu je ihned zavolán request `SelfServiceDataRequest`. Ten vrací užitečná data o modulu, především jeho metadata a další data, která jsou potřebná pro funkcionalitu Self Service. Z těchto dat jsou dále získávány všechny požadované hodnoty pro následující dropdowny a textová pole.

Source Snapshot (zdrojový snapshot) je verze zálohy zdrojového modulu. Snapshoty zdrojového modulu se načítají z jeho metadat. Obvykle se jedná o „Live Database“ (aktuální data v databázi) a dále jednotlivé snapshoty, které byly vytvořeny buď v rámci automatického zálohování, nebo ručně uživatelem.

Period type je možno zvolit ze tří hodnot - HISTORICAL, FORECAST a ALL. Zvolená hodnota říká, jestli se mají z modulu přenést historická, forecast nebo všechna data. Toto označení se odvíjí od aktuálního časového období (obvykle kvartálu) nastaveného na modulu. Všechna data před tímto obdobím a během něj se považují za historická a data pocházející z budoucnosti jsou typu forecast.

Název transformace může uživatel buď napsat sám, nebo použít výchozí název, který je automaticky vytvořen z názvů zdrojového a cílového modulu.

Komentář je nepovinný text, který může například doplnit informace o transformaci.

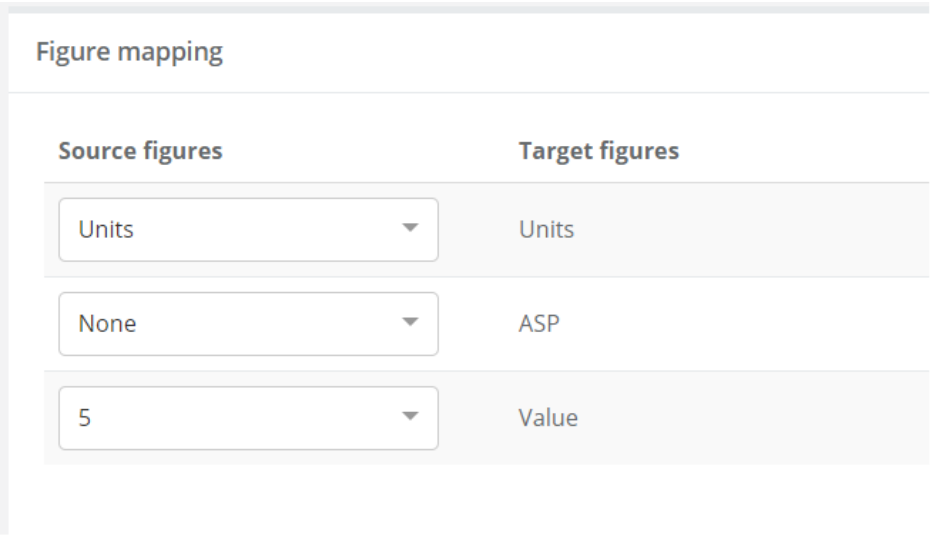


Obrázek 3: Sekce General info

Dalším úkolem bylo přidat sekci pro Figure mapping. V této sekci se na číselné hodnoty (například Units - počet prodaných kusů) ze zdrojového modulu mapují figures z modulu cílového. Toto mapování tedy říká do kterých číselných atributů se hodnoty ze zdrojového modulu vloží.

V této sekci byla za tímto účelem připravena tabulka se dvěma sloupci. V pravém sloupci jsou vypsány figures z cílového modulu. V levém sloupci je pak na každém řádku dropdown obsahující figures z modulu zdrojového, viz obrázek 4. Uživatel má 3 možnosti:

1. nechat dropdown prázdný, nic z něj nevybrat. V takovém případě se figure vůbec nepoužije.
2. vybrat v dropdownu jednu z figures existujících na zdrojovém modulu. Při transformaci se pak provede mapování mezi vybranými figures.
3. vložit do dropdownu novou číselnou hodnotu a vybrat ji. Tato hodnota se pak použije jako konstanta na všech řádcích.



Obrázek 4: Sekce Figure mapping

Byla vytvořena mapa, ve které figures z cílového modulu představují klíče. Hodnoty jsou na začátku nastaveny na konstantu NO_FIGURE_MAPPING, která říká, že pro tuto cílovou figure nebyla žádná hodnota zvolena. Při jakékoli změně v dropdownu se spustí event, který nastaví hodnotu pro požadovaný klíč na právě zvolený prvek.

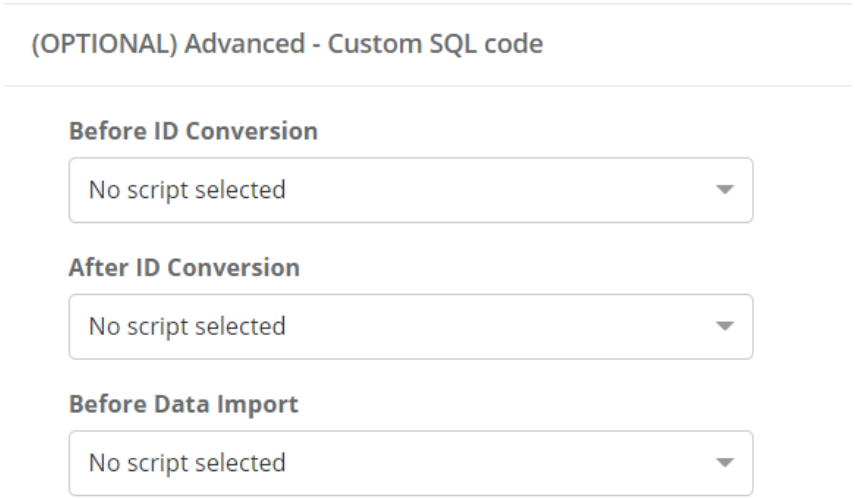
Použitá komponenta ng-select má property addTag, která byla využita pro přidávání konstantních hodnot. Pokud uživatel začne psát na vstup, nabídne se mu automaticky v dropdownu možnost pro přidání konstantní hodnoty. Tato volba je však povolena pouze v případě, že má vstup číselnou hodnotu.

Po kliknutí na potvrzovací tlačítko Save Transformation proběhne vložení všech nastavených hodnot do seznamu figures na objektu transformace, viz výpis 7. Součástí je také validace, že je alespoň jedna figure namapovaná.

```
transformation.figures = [];
for (let entry of this.figureMapping.mappedFigures.entries()) {
   if (Number(entry[1])) { //jedná se o konstantu
      let transFigure = new RemoteTransFigureConstant();
      transFigure.targetFigure = entry[0].id;
      transFigure.constantValue = +entry[1];
      transformation.figures.push(transFigure);
   } else if (entry[1] !== this.figureMapping.NO_FIGURE_MAPPING) {
      let transFigure = new RemoteTransFigureSimple();
      transFigure.targetFigure = entry[0].id;
      transFigure.sourceFigure = entry[1].id;
      transformation.figures.push(transFigure);
   }
}
```

Výpis 7: Mapování figures

Poslední sekcí, kterou jsem měla v rámci vytváření transformací připravit, byla sekce pro výběr sql skriptů, viz obrázek 5. Uživatel má v této sekci možnost vybrat si některou z existujících databázových procedur pro použití před konverzí ID (během které dochází k překládání názvů atributů na jejich ID), po konverzi ID nebo před importem dat. Tato možnost je pro většinu uživatelů příliš pokročilá a využívá se zřídka, proto je výslovně označena jako Optional (volitelná).



Obrázek 5: Sekce Custom SQL code

Pro získání procedur daného modulu byl vytvořen nový request. V handleru tohoto requestu se provádí dotaz nad databází uvedený ve výpisu 8.

```
SELECT routines.specific_schema || '.' || routines.routine_name
FROM information_schema.routines
LEFT JOIN information_schema.parameters ON routines.specific_name=parameters
    .specific_name
WHERE routines.specific_schema='{}'
ORDER BY routines.routine_name, parameters.ordinal_position
```

Výpis 8: Získání procedur daného schématu z databáze

Při provádění dotazu je pak vloženo schéma konkrétního modulu místo složených závorek. Ze získaných názvů procedur dochází k odstranění těch, které se týkají derivovaných atributů, jelikož ty slouží právě k odvození hodnot těchto atributů a tudíž nejsou relevantní. Tyto vyfiltrované procedury se potom zasílají zpět do Ferdy Admina v serverové odpovědi.

Na stránku s vytvářením transformací byly přidány 3 dropdowny pro každý typ použití. Výchozí hodnotou je nápis informující o tom, že nebyl vybrán žádný skript. Pokud uživatel nějaký vybere, volba se uloží do nastavení transformace při kliknutí na tlačítko Save.

Kromě vytváření transformace bylo potřeba naimplementovat také jejich editaci a mazání. K tomuto účelu byly přidány ikony znázorňující jednotlivé akce ke každé transformaci v seznamu. Pro mazání je vytvořena metoda, která v parametru přijímá objekt transformace, kterou si uživatel přeje smazat. Nejprve se pomocí alertu z knihovny sweet alert uživateli zobrazí potvrzovací dialog, zda si opravdu přeje danou transformaci smazat. Po potvrzení je v metodě vytvořena instance objektu DeleteRequest a na server se zašle požadavek na smazání transformace s daným ID. Následně se uživateli zobrazí okno s informací o úspěchu (případně neúspěchu) mazání a znovu se načte seznam transformací.

Princip editace transformace je takový, že po kliknutí na ikonu znázorňující úpravu se uživateli znovu otevře stejné okno, jako při vytváření transformace, ale veškeré nastavení je již předvyplněno a uživatel může požadované hodnoty měnit. Pro umožnění této funkcionality se předává do vytvářecí komponenty objekt upravované transformace pomocí dekorátoru `@Input()`. Pokud má tento objekt hodnotu null, znamená to, že uživatel vytváří novou transformaci. V opačném případě se vlastnosti tohoto objektu použijí jako ngModely v jednotlivých inputech. Editace byla rovněž implementována pouze u HLC transformací, u ostatních typů tato možnost zatím není podporována.

4.6 Monitorování místa na disku

Cílem bylo usnadnit kontrolování volného místa na serverových discích a zajistit včasné varování v případě, že místo dochází. Nejprve bylo potřeba přidat nový check do Ferda Monitoru. Tento check měl kontrolovat obsazené místo na Ferda serverech. Na těchto serverech běží aplikace Ferda, a také jsou tam obsažena data, která tam posílají uživatelé, například vyexportované

soubory dostupné ke stažení, caches, různé dočasné soubory apod. Jelikož se jedná o unixové servery, bylo získávání obsazeného místa na disku zajištěno pomocí příkazu df -h, které vrací obsazenost disku v čitelném formátu. Získané číslo se nyní periodicky zasílá každých 60 sekund do Ferdy Monitoru spolu s dalšími informacemi, které monitor sleduje. V případě, že dojde k využití místa na disku na více než 80%, Ferda Monitor zobrazí hlášku informující o překročení limitu. Zároveň byla přidána informace o obsazeném místu na disku v procentech do tabulky ke každému serverovému nodu, jak je možno vidět na obrázku 6, sloupec **Disk space utilization**.

Ferda server instances ✔

ID	Last Update	Disk space utilization	State
prod_node_1	Wed Apr 03 03:03:11 EDT 2019	40%	OK
prod_node_2	Wed Apr 03 03:02:39 EDT 2019	40%	OK
prod_node_3	Wed Apr 03 03:02:18 EDT 2019	40%	OK
prod_node_4	Wed Apr 03 03:02:27 EDT 2019	40%	OK
prod_node_5	Wed Apr 03 03:02:20 EDT 2019	40%	OK
prod_node_6	Wed Apr 03 03:02:23 EDT 2019	40%	OK

Obrázek 6: Serverové instance a jejich využití místa na disku

Pro kontrolu místa na databázových serverech, čili serverech, na kterých jsou uložena všechna data a metadata, byl vytvořen skript v Pythonu. Ten opět pomocí příkazu df zjišťuje velikosti jednotlivých diskových oddílů na těchto serverech. Získané informace pak ukládá do tabulky do nově vytvořeného databázového schématu monitor. Tento skript je spouštěn v pravidelných intervalech pomocí cron jobu. Do Ferdy Monitoru byl přidán databázový check, který kontroluje tabulkové záznamy vytvořené tímto skriptem. V případě, že dojde k překročení hranice, ohlásí Ferda Monitor varování.

Aby byli vývojáři včas informováni o překročeném limitu nebo jiných problémech s aplikací Ferda, bylo zavedeno také automatické vytváření JIRA ticketů s maximální prioritou (blocker). Zároveň byla do Ferdy Monitoru přidána sekce Unresolved Issues, která obsahuje seznam aktuálně otevřených ticketů vztahujících se k problémům z monitoru. V případě, že žádné otevřené tickety neexistují, je sekce zbarvená do zelena, v opačném případě je oranžová, viz obrázek 7.

▼ Unresolved Ferda Monitor issues ⚠

Issue	Summary
FSU-13907	FERDA MONITOR - Database check failed: Last daily backup failed.

▼ Unresolved Ferda Monitor issues ✔

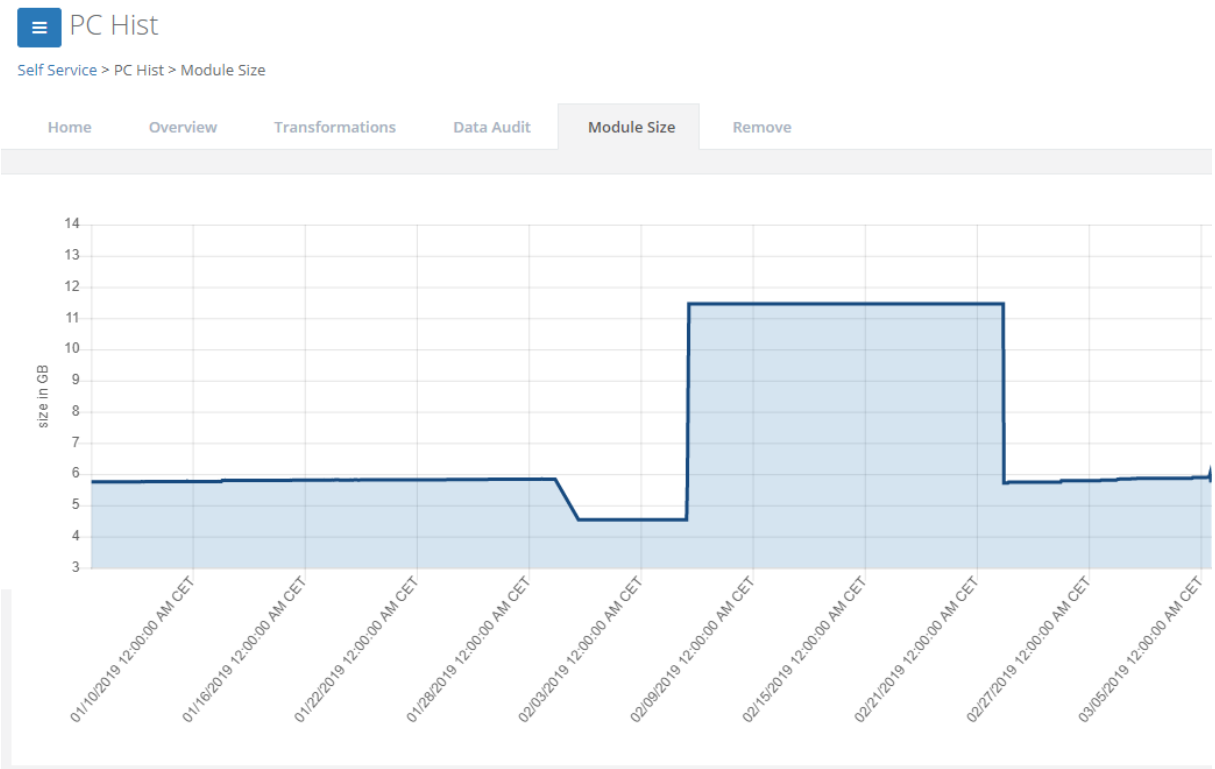
Issue
No unresolved issues.

Obrázek 7: Unresolved issues - otevřené tickety ve Ferda Monitor

Kromě obsazení místa na serverech bylo také třeba monitorovat velikost jednotlivých modulů. V rámci tohoto úkolu bylo především potřeba stanovit si, v jakém případě bude nárůst velikosti modulu považován za problém, jelikož některé moduly jsou skutečně velké (to se týká především forecast modulů), zatímco jiné moduly oproti nim zabírají pouze zlomek místa. Byl proto vytvořen databázový check (SQL dotaz, který se periodicky spouští), který pro každý modul zjišťuje, zda během posledních 6 hodin nedošlo k nárustu jeho velikosti více než o 30% původní velikosti a zároveň o více než 1 GB. Díky tomu se eliminovaly plané poplachy, které by mohly vznikat při následujících případech:

1. velmi malý modul vzrostl objemově například o polovinu, nicméně jedná se stále o velmi malou kapacitu, kterou není potřeba považovat za problém.
2. velmi objemný modul narostl o více než 1 GB, ale vzhledem k jeho celkové velikosti je tento nárust stále zanedbatelný.

Zmíněný SQL dotaz se spouští nad nově vytvořenou databázovou tabulkou, která obsahuje informace o velikostech jednotlivých modulů. Do této tabulky se ukládají informace pomocí nově vytvořeného requestu `StoreTechnologiesSizeRequest`, který je volán každou hodinu pomocí Cron jobu.



Obrázek 8: Graf velikosti modulu

Pro lepší vizualizaci byla do Ferdy Admina přidána k modulům záložka Module Size (velikost modulu), která zobrazuje graf znázorňující změny velikosti modulu v průběhu času. Příklad takového grafu je zobrazen na obrázku 8.

V současné době je pro vývojáře jednodušší odhalit příčinu docházejícího místa na serverových discích. Pokud dojde k zaplnění disku, mohou spustit tzv. *vaccum*, které uvolní prostor v datových souborech, které jsou zaplněné neplatnými záznamy. Postgres totiž při příkazech UPDATE a DELETE jednotlivé záznamy neodstraňuje, protože se jedná o příliš náročnou operaci, ale pouze je označí za neplatné. Záznamy tak v databázi stále zůstávají a vacuum slouží k tomu, aby je pročistilo.

5 Závěr

Cílem této práce bylo přiblížit zaměření firmy IDC a popsat průběh mé praxe v této firmě. Nejprve jsem se krátce věnovala představení firmy a také týmu, jehož jsem byla součástí. Poté jsem vysvětlila, k čemu slouží klíčové projekty Ferda, Ferda Monitor a Ferda Admin. Nakonec jsem rozepsala jednotlivé úkoly, na kterých jsem pracovala, naznačila jsem postup při jejich řešení a také jsem vysvětlila, proč byla daná řešení přínosná pro uživatele. Pro znázornění a lepší představu jsem uvedla také výpisy některých zdrojových kódů nebo obrázky z dané aplikace zobrazující nově přidané funkcionality.

5.1 Uplatněné znalosti a dovednosti

Během praxe jsem využila mnoho dovedností, které jsem získala v průběhu studia na vysoké škole. Jednalo se především o znalosti z předmětu Programovací jazyky I, ve kterém jsem se poprvé blíže setkala s programovacím jazykem Java. Díky tomuto předmětu jsem si tento programovací jazyk oblíbila a rozhodla jsem se proto hledat si bakalářskou praxi na pozici programátora v Javě. Klíčové byly nepochybně také předměty Programování I a II a Algoritmy I a II, které mě naučily základní programovací znalosti a přemýšlení.

Podstatné databázové znalosti jsem nabyla v předmětech Úvod do databázových systémů a Databázové a informační systémy, které mě obeznámily se základními pojmy databázových technologií, naučily mě používat jazyk SQL pro definici, manipulaci a dotazování dat a pomohly mi pochopit jak fungují databázové funkce a procedury. I to pro mě bylo velmi důležité, jelikož jsem během své práce využívala databázi téměř denně.

5.2 Nabyté znalosti a dovednosti

Práce mi poskytla možnost jednak uplatnit znalosti, které jsem získala během studia, ale především nabýt nové, jelikož reálné prostředí vývoje se od výuky velice liší a vyžaduje schopnost propojovat technologie, se kterými jsem se doposud setkala odděleně. Během vykonávání praxe jsem zdokonalila svou znalost programovacího jazyka Java, vyzkoušela jsem si pokročilejší práci s databází a naučila jsem se používat pro mne zcela nový framework, Angular. Zároveň jsem získala kompletní představu o fungování serveru a vytváření a zpracovávání požadavků. Získala jsem také praktické zkušenosti s fungováním verzovacího systému Git, se kterým jsem se do té doby nesetkala. Tato znalost je velmi podstatná, protože se jedná o nezbytnou součást vývoje všech větších projektů. Také jsem pochopila, že je podstatné se umět orientovat v cizím kódu a umět ho znovu využít.

5.3 Shrnutí

Absolvování bakalářské praxe považuji za cennou zkušenost a hodnotím ji velice kladně. Měla jsem možnost pracovat ve skvělém kolektivu zkušených programátorů a velmi příjemném pro-

středí. Díky této praxi jsem získala mnoho nových technických dovedností, ale také pohled na to, jak probíhá vývoj software ve firmě. Zjistila jsem, že programování je jen jeden z mnoha aspektů, které musí programátor ovládat, a že komunikace v týmu je také velmi důležitá. Může se například stát, že implementace zadaného úkolu by s sebou přinesla nezamýšlené potenciální rizika kódu nebo špatné programovací praktiky. Proto je dobré se občas nad aktuálním vývojem objektivně zamyslet a při nalezení některé z takovýchto skutečností nedělat další kroky, ale prodiskutovat ji se zkušenějšími kolegy, kteří problematice rozumí. Věřím, že mi tato skutečnost pomůže v mém dalším uplatnění na trhu práce.

Literatura

[1] idc-cema.com. (2019). Company Overview | About IDC | IDC CEMA. [online] Available at: https://idc-cema.com/eng/about-idc/company-overview [Accessed 21 Mar. 2019].

[2] Tiobe.com. (2019). TIOBE Index | TIOBE - The Software Quality Company. [online] Available at: https://www.tiobe.com/tiobe-index/ [Accessed 7 Mar. 2019].

[3] Oracle.com. (2019). The Java Language Environment. [online] Available at: https://www.oracle.com/technetwork/java/intro-141325.html [Accessed 7 Mar. 2019].

[4] Angular.io. (2019). Angular. [online] Available at: https://angular.io/docs [Accessed 12 Mar. 2019].

[5] SitePoint. (2019). AngularJS and Angular 2+: a Detailed Comparison — SitePoint. [online] Available at: https://www.sitepoint.com/angularjs-vs-angular/ [Accessed 12 Mar. 2019].

[6] Postgresql.org. (2019). PostgreSQL: About. [online] Available at: https://www.postgresql.org/about/ [Accessed 12 Mar. 2019].

[7] Postgresqltutorial.com. (2019). What is PostgreSQL. [online] Available at: http://www.postgresqltutorial.com/what-is-postgresql/ [Accessed 12 Mar. 2019].

[8] Git-scm.com. (2019). About - Git. [online] Available at: https://git-scm.com/about [Accessed 3 Mar. 2019].

[9] Git-scm.com. (2019). Git - A Short History of Git. [online] Available at: https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git [Accessed 3 Mar. 2019].